

TUGAS BESAR 2

IF2123 Aljabar Linier dan Geometri

Aplikasi Nilai Eigen dan Vektor Eigen dalam Kompresi Gambar



Dipersiapkan oleh:

Kelompok 8 (ALGEOGRAFI)

Nadia Mareta Putri Leiden	13520007
---------------------------	----------

Taufan Fajarama Putrawansyah R	13520031
--------------------------------	----------

Raden Haryosatyo Wisjnunandono	13520070
--------------------------------	----------

**PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2021**

DAFTAR ISI

DAFTAR ISI	1
BAB 1: DESKRIPSI MASALAH	2
BAB 2: TEORI SINGKAT	2
2.1 Perkalian Matriks	2
2.2 Nilai Eigen	2
2.3 Vektor Eigen	3
2.4 Matriks SVD (Singular Value Decomposition)	3
BAB 3: IMPLEMENTASI	4
3.1 Front End	4
3.2 Back End	4
3.3 Algoritma SVD (Singular Value Decomposition)	7
BAB 4: EKSPERIMEN	16
BAB 5: PENUTUP	19
5.1 Kesimpulan	19
5.2 Saran	19
5.3 Refleksi	19
REFERENSI	20

BAB 1: DESKRIPSI MASALAH

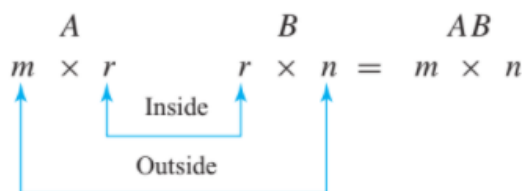
Gambar adalah suatu hal yang sangat dibutuhkan pada dunia modern ini. Kita seringkali berinteraksi dengan gambar baik untuk mendapatkan informasi maupun sebagai hiburan. Gambar digital banyak sekali dipertukarkan di dunia digital melalui file-file yang mengandung gambar tersebut. Seringkali dalam transmisi dan penyimpanan gambar ditemukan masalah karena ukuran file gambar digital yang cenderung besar.

Kompresi gambar merupakan suatu tipe kompresi data yang dilakukan pada gambar digital. Dengan kompresi gambar, suatu file gambar digital dapat dikurangi ukuran filenya dengan baik tanpa mempengaruhi kualitas gambar secara signifikan. Terdapat berbagai metode dan algoritma yang digunakan untuk kompresi gambar pada zaman modern ini. Salah satu algoritma yang dapat digunakan untuk kompresi gambar adalah algoritma SVD (Singular Value Decomposition). Pada kesempatan kali ini, kami mendapatkan tantangan untuk membuat website kompresi gambar sederhana dengan menggunakan algoritma SVD.

BAB 2: TEORI SINGKAT

2.1 Perkalian Matriks

Definisi perkalian matriks mensyaratkan bahwa jumlah kolom matriks pertama A sama dengan jumlah baris matriks kedua B untuk membentuk produk AB. Jika kondisi ini tidak terpenuhi, produk tidak terdefinisi. Cara mudah untuk menentukan apakah produk dari dua matriks terdefinisi adalah dengan menuliskan ukuran matriks pertama dan di sebelah kanannya, tuliskan ukuran matriks kedua. Jika angka-angka di dalamnya sama, maka hasil kali terdefinisi dan angka-angka di luar memberikan ukuran produk.



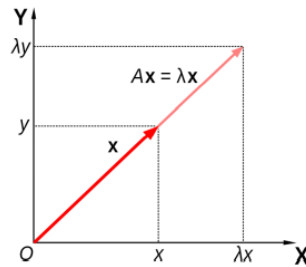
2.2 Nilai Eigen

Nilai Eigen adalah nilai karakteristik dari matriks yang berukuran $n \times n$. Karena Eigen berasal dari bahasa Jerman yang berarti "asli" atau "karakteristik". Penggunaan nilai Eigen ini akan sangat mempengaruhi pada penentuan Vektor Eigen. Adapun, persamaan nilai Eigen yang dapat dipakai adalah seperti di bawah ini.

$$Ax = \lambda x$$

A menyatakan vektor yang ditentukan, x menyatakan vektor eigen, dan λ menyatakan nilai eigen. Nilai – nilai eigen ini bisa bernilai 1 angka atau 2 angka tergantung dengan vektor

A. Operasi ini menyebabkan vektor Eigen menyusut ataupun memanjang bergantung dengan nilai eigen yang digunakan seperti pada ilustrasi di bawah.



2.3 Vektor Eigen

Vektor Eigen menyatakan sebuah vektor kolom pada formula yang sudah diberikan pada poin 2.2, yang apabila vektor ini dikalikan dengan vektorn $n \times n$ akan menghasilkan vektor yang merupakan kelipatan dari vektor itu sendiri. Sehingga setelah mendapatkan nilai-nilai Eigen, dapat dicari Vektor Eigen melalui formula di bawah ini:

$$\lambda I - Ax = 0$$

Kemudian, untuk mencari terlebih dahulu nilai karakteristik nya dapat digunakan formula:

$$\det(\lambda I - Ax) = 0$$

2.4 Matriks SVD (Singular Value Decomposition)

Matriks SVD adalah teorema dalam aljabar linier yang menyatakan bahwa sebuah matriks dua dimensi dapat dipecah menjadi hasil perkalian dari 3 sub-matriks yaitu

matriks ortogonal U, matriks diagonal S, dan transpose dari matriks ortogonal V. Dekomposisi matriks ini dapat dinyatakan sesuai persamaan berikut.

$$A_{m \times n} = U_{m \times m} S_{m \times n} V_{n \times n}^T$$

Matriks U adalah matriks yang kolomnya terdiri dari vektor eigen ortonormal dari matriks AA^T . Matriks ini menyimpan informasi yang penting terkait baris-baris matriks awal, dengan informasi terpenting disimpan di dalam kolom pertama. Matriks S adalah matriks diagonal yang berisi akar dari nilai eigen matriks U atau V yang terurut menurun. Matriks V adalah matriks yang kolomnya terdiri dari vektor eigen ortonormal dari matriks $A^T A$. Matriks ini menyimpan informasi yang penting terkait kolom-kolom matriks awal, dengan informasi terpenting disimpan dalam baris pertama.



BAB 3: IMPLEMENTASI

3.1 Front End

Tech stack: Flask

Menggunakan Flask dengan format penulisan yang terletak pada directory template. File yang digunakan ber-ekstensi html. Pada file ini terdiri dari tata letak tombol submit, input tingkat kompresi diatur dengan minimum 0 sampai 100. Untuk bisa menyatukan front-end dengan back-end kami menggunakan `render_template` yang diimport pada back-end.

```
1 <html>
2 <head>
3 <title>Kompresi Foto Menggunakan SVD</title>
4 <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" />
5 <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.0/jquery.min.js"></script>
6 </head>
7 <body>
8 <p><h1 align="center">Kompresi Foto Menggunakan SVD</h1></p>
9 <div class="container">
10 <div class="row">
11 <br>
12 <h2>Pilih file yang ingin di-upload</h2>
13 <h5>Perhatian: file anda akan secara otomatis terkompresi, anda akan di-direct ke tab berisi foto yang sudah dikompresi.</h5>
14 <p>
15     {% with messages = get_flashed_messages() %}
16     {% if messages %}
17     <ul>
18     {% for message in messages %}
19     <li>{{ message }}</li>
20     {% endfor %}
21     </ul>
22     {% endif %}
23     {% endwith %}
24 </p>
25 <br>
26 <form method="post" action="/" enctype="multipart/form-data">
27 <dl>
28 <p>
29 <label for="const">Masukkan konstanta:</label>
30 <input type="number" id="const" name="const" min="0" max="100">
31 <br>
32 <input type="file" name="file" class="form-control" autocomplete="off" required>
33 </p>
34 </dl>
35 <p>
36 <input type="submit" value="Submit" class="btn btn-info">
37 </p>
38 </form>
39 </p>
40 <p>
41 <input type="submit" value="Submit" class="btn btn-info">
42 </p>
43 </div>
44 </div>
45 </div>
46 </body>
47 </html>
```

3.2 Back End

Tech stack: Flask

Pada bagian ini berisikan routing-routing yang dibutuhkan pada program, termasuk untuk mengolah input user, menyimpan file ke directory, dan mengatur url untuk download file. Pada bagian backend ini juga terdapat algoritma SVD yang akan digunakan dalam pengolahan foto, yang nantinya akan lebih jauh dibahas pada bagian selanjutnya. Untuk

bisa menggunakan fitur routing tersebut kami mengimport beberapa fitur bawaan dari flask untuk digunakan antara lain: `render_template`, `url_for`, `redirect`, `send_from_directory`, dan masih banyak lainnya.

Supaya program dapat berjalan user harus mengganti variabel `UPLOAD_FOLDER` ke path `../api/uploads` pada directory masing-masing.

```
app.py > upload_image
1  from flask import Flask, flash, request, redirect, url_for, render_template, send_from_directory
2  import urllib.request
3  import os
4  from werkzeug.utils import secure_filename
5
6  from PIL import Image
7  import numpy as np
8  from numpy.linalg import norm
9  from math import sqrt
10 from random import normalvariate
11 import time
12
13
14 app = Flask(__name__)
15
16 UPLOAD_FOLDER = 'D:/algeodum/api/uploads'
17
18 app.secret_key = "secret key"
19 app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
20
21 ALLOWED_EXTENSIONS = set(['png', 'jpg', 'jpeg'])
22
23 def randomUnitVector(n):
24     unnormalized = [normalvariate(0, 1) for _ in range(n)]
25     theNorm = sqrt(sum(x * x for x in unnormalized))
26     return [x / theNorm for x in unnormalized]
27
28
29 def svd_1D(A, epsilon=1e-10):
30     n, m = A.shape
31     x = randomUnitVector(min(n,m))
32     lastV = None
33     currentV = x
34
35     if n > m:
36         B = np.dot(A.T, A)
37     else:
38         B = np.dot(A, A.T)
```

```

def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/', methods=['GET', 'POST'])
def upload_image():
    if 'file' not in request.files:
        flash('Tidak ada file.')
        return redirect(request.url)
    file = request.files['file']
    constant = request.form.get('cons', type=int)
    if file.filename == '':
        flash('Tidak ada foto yang Anda pilih.')
        return redirect(request.url)
    if constant == 0:
        flash('Masukkan angka yang benar, yaitu lebih dari 0.')
        return redirect(request.url)
    if file and allowed_file(file.filename):

        # pemrosesan file dilakukan disini

        #penghitungan waktu
        starttime = time.time()

        #file extension
        if(file.filename.rsplit('.', 1)[1].lower() == 'jpg'):
            file = imgcompres(file.filename, constant)
            filename = "result.jpg"
        elif(file.filename.rsplit('.', 1)[1].lower() == 'png'):
            file = imgcompres(file.filename, constant)
            filename = "result.png"
        else:
            file = imgcompres(file.filename, constant)
            filename = "result.jpeg"

```

```

195         else:
196             file = imgcompres(file.filename, constant)
197             filename = "result.jpeg"
198
199             end = time.time()
200
201             flash(f"Image compression time is {end - starttime} s")
202             file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
203             flash('Foto Anda telah berhasil di upload')
204             return redirect(url_for('download_file', name=filename))
205         else:
206             flash('Anda hanya boleh mengupload jpg, jpeg atau png.')
207             return redirect(request.url)
208
209     @app.route('/uploads/<name>')
210     def download_file(name):
211         return send_from_directory(app.config["UPLOAD_FOLDER"], name)
212
213
214 if __name__ == "__main__":
215     app.run()

```

3.3 Algoritma SVD (Singular Value Decomposition)

Tech stack: Python

Pada bagian ini berisi tentang algoritma yang digunakan untuk mengkompresi gambar dengan menggunakan singular value decomposition. Pada bagian ini dijelaskan tentang algoritma SVD yang dipakai yaitu randomized SVD dan Power Iteration. Algoritma Power Iteration memungkinkan kita untuk mendapatkan vektor eigen dan nilai eigen dari yang terbesar. Dari hasil power iteration itu kita dapat menyusun SVD. Randomized SVD digunakan untuk mempercepat algoritma dengan cara mengecilkan terlebih dahulu matrix yang ingin didekomposisi dengan power iteration.

Untuk kompresi gambar, awalnya gambar diubah ke dalam bentuk matriks. Karena gambar berwarna merupakan array dimensi, maka mula-mula gambar dipisah dulu menjadi 3 bagian (R, G, B). Masing-masing dari array 2 dimensi R, G, B tersebut kemudian didekomposisi dengan SVD yang terdapat pada paragraf sebelumnya. Setelah didekomposisi, ketiga array 2 dimensi tersebut disatukan kembali.

I. Import Library yang ingin digunakan


```

from PIL import Image
import numpy as np
from numpy.linalg import norm
from math import sqrt
from random import normalvariate
import time

```

II. Algoritma SVD dengan cara Power Iteration

```

def randomUnitVector(n):
    unnormalized = [normalvariate(0, 1) for _ in range(n)]
    theNorm = sqrt(sum(x * x for x in unnormalized))
    return [x / theNorm for x in unnormalized]

def svd_1D(A, epsilon=1e-10):
    n, m = A.shape
    x = randomUnitVector(min(n,m))
    lastV = None
    currentV = x

    if n > m:
        B = np.dot(A.T, A)
    else:
        B = np.dot(A, A.T)

    iterations = 0
    #Lakukan iterasi sampai dengan nilai vektor hasil iterasi terakhir - iterasi sebelumnya
    # mencapai epsilon (sudah konvergen ke suatu nilai, i.e eigenvektor terbesar)
    while True:
        iterations += 1
        lastV = currentV
        currentV = np.dot(B, lastV)
        currentV = currentV / norm(currentV)

        if abs(np.dot(currentV, lastV)) > 1 - epsilon:
            return currentV

```

```

def svd(A, k=None, epsilon=1e-10):
    A = np.array(A, dtype=float)
    n, m = A.shape
    svdSoFar = []
    if k is None:
        k = min(n, m)

    for i in range(k):
        matrixFor1D = A.copy()

        for singularValue, u, v in svdSoFar[:i]:
            matrixFor1D -= singularValue * np.outer(u, v)

        if n > m:
            v = svd_1D(matrixFor1D, epsilon=epsilon) # vektor singular selanjutnya
            u_unnormalized = np.dot(A, v)
            sigma = norm(u_unnormalized) # singular value selanjutnya
            u = u_unnormalized / sigma
        else:
            u = svd_1D(matrixFor1D, epsilon=epsilon) # vektor singular selanjutnya
            v_unnormalized = np.dot(A.T, u)
            sigma = norm(v_unnormalized) # singular value selanjutnya
            v = v_unnormalized / sigma

        svdSoFar.append((sigma, u, v))

    singularValues, us, vs = [np.array(x) for x in zip(*svdSoFar)]
    return us.T, singularValues, vs

```

III. Algoritma Randomized SVD untuk mempercepat runtime program

```

def randomSVD(X, r, q, p):
    #FASE 1: dekomposisi matrix X dengan cara sampling kolom X dengan random oleh matrix P
    ny = X.shape[1]
    P = np.random.randn(ny, r+p)
    Z = X @ P
    for k in range(q):
        Z = X @ (X.T @ Z)

    Q, R = np.linalg.qr(Z, mode='reduced')
    #FASE 2: Menghitung SVD dari matrix Y
    Y = Q.T @ X
    UY, S, VT = svd(Y)
    U = Q @ UY

    return U, S, VT

```

IV. Main program untuk kompresi gambar

```

def imgcompres(x, k):
    img = np.asarray(Image.open(x))
    m = img.shape[0]
    n = img.shape[1]

    # array RGB
    r = img[:, :, 0]
    g = img[:, :, 1]
    b = img[:, :, 2]

    print("compressing...")
    starttime = time.time()

    q = 1
    p = 5

    Ur, Sr, VTr = randomSVD(r, k, q, p)
    Ug, Sg, VTg = randomSVD(g, k, q, p)
    Ub, Sb, VTb = randomSVD(b, k, q, p)

    k = round(((100-k)*m*n)/(100*(m+1+n)))

    # menggabungkan tiap komponen svd sebanyak k kolom/baris
    rr = np.dot(Ur[:, :k], np.dot(np.diag(Sr[:k]), VTr[:k, :]))
    rg = np.dot(Ug[:, :k], np.dot(np.diag(Sg[:k]), VTg[:k, :]))
    rb = np.dot(Ub[:, :k], np.dot(np.diag(Sb[:k]), VTb[:k, :]))

    print("arranging...")

```

```

# buat array kosong untuk menampung matriks gambar kompresi
rimg = np.zeros(img.shape)

# menggabungkan tiap channel RGB
rimg[:, :, 0] = rr
rimg[:, :, 1] = rg
rimg[:, :, 2] = rb

# konversi angka matriks
for ind1, row in enumerate(rimg):
    for ind2, col in enumerate(row):
        for ind3, value in enumerate(col):
            if value < 0:
                rimg[ind1, ind2, ind3] = abs(value)
            if value > 255:
                rimg[ind1, ind2, ind3] = 255

# konversi gambar & save file
compressed_image = rimg.astype(np.uint8)
compressed_image = Image.fromarray(compressed_image)

end = time.time()

# total time taken
print(f"Runtime of the program is {end - starttime}")

return compressed_image

```

```

from PIL import Image
import numpy as np
from numpy.linalg import norm
from math import sqrt
from random import normalvariate
import time

app = Flask(__name__)

UPLOAD_FOLDER = 'D:/algeodum/api/uploads'

app.secret_key = "secret key"
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

ALLOWED_EXTENSIONS = set(['png', 'jpg', 'jpeg'])

def randomUnitVector(n):
    unnormalized = [normalvariate(0, 1) for _ in range(n)]
    theNorm = sqrt(sum(x * x for x in unnormalized))
    return [x / theNorm for x in unnormalized]

def svd_1D(A, epsilon=1e-10):
    n, m = A.shape
    x = randomUnitVector(min(n,m))
    lastV = None
    currentV = x

    if n > m:
        B = np.dot(A.T, A)
    else:
        B = np.dot(A, A.T)

```

```

57     else:
58         B = np.dot(A, A.T)
59
60     iterations = 0
61     #lakukan iterasi sampai dengan nilai vektor hasil iterasi terakhir - iterasi sebelumnya
62     # mencapai epsilon (sudah konvergen ke suatu nilai, i.e eigenvektor terbesar)
63     while True:
64         iterations += 1
65         lastV = currentV
66         currentV = np.dot(B, lastV)
67         currentV = currentV / norm(currentV)
68
69         if abs(np.dot(currentV, lastV)) > 1 - epsilon:
70             return currentV
71
72
73 def svd(A, k=None, epsilon=1e-10):
74     A = np.array(A, dtype=float)
75     n, m = A.shape
76     svdSoFar = []
77     if k is None:
78         k = min(n, m)
79
80     for i in range(k):
81         matrixFor1D = A.copy()
82
83         for singularValue, u, v in svdSoFar[:i]:
84             matrixFor1D -= singularValue * np.outer(u, v)
85
86         if n > m:
87             v = svd_1D(matrixFor1D, epsilon=epsilon) # vektor singular selanjutnya
88             u_unnormalized = np.dot(A, v)
89             sigma = norm(u_unnormalized) # singular value selanjutnya
90             u = u_unnormalized / sigma
91         else:
92             u = svd_1D(matrixFor1D, epsilon=epsilon) # vektor singular selanjutnya
93             v_unnormalized = np.dot(A.T, u)
94             sigma = norm(v_unnormalized) # singular value selanjutnya

```

```

74         sigma = norm(v_unnormalized) # singular value selanjutnya
75         v = v_unnormalized / sigma
76
77         svdSoFar.append((sigma, u, v))
78
79     singularValues, us, vs = [np.array(x) for x in zip(*svdSoFar)]
80     return us.T, singularValues, vs
81
82
83 def randomSVD(X,r,q,p):
84     #FASE 1: dekomposisi matrix X dengan cara sampling kolom X dengan random oleh matrix P
85     ny = X.shape[1]
86     P = np.random.randn(ny,r+p)
87     Z = X @ P
88     for k in range(q):
89         Z = X @ (X.T @ Z)
90
91     Q, R = np.linalg.qr(Z,mode='reduced')
92     #FASE 2: Menghitung SVD dari matrix Y
93     Y = Q.T @ X
94     UY, S, VT = svd(Y)
95     U = Q @ UY
96
97     return U, S, VT
98
99 def imgcompres(x, k):
100     img = np.asarray(Image.open(x))
101     m = img.shape[0]
102     n = img.shape[1]
103
104     # array RGB
105     r = img[:, :, 0]
106     g = img[:, :, 1]
107     b = img[:, :, 2]
108
109     print("compressing...")
110     starttime = time.time()
111


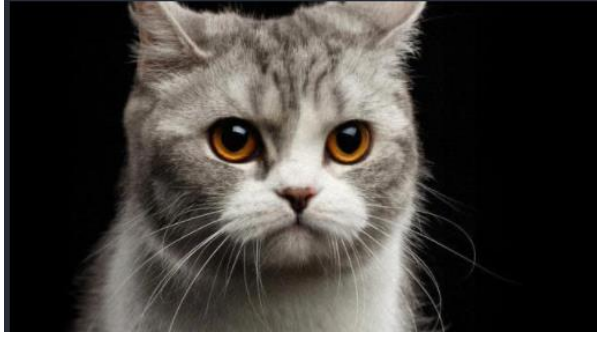

```




```



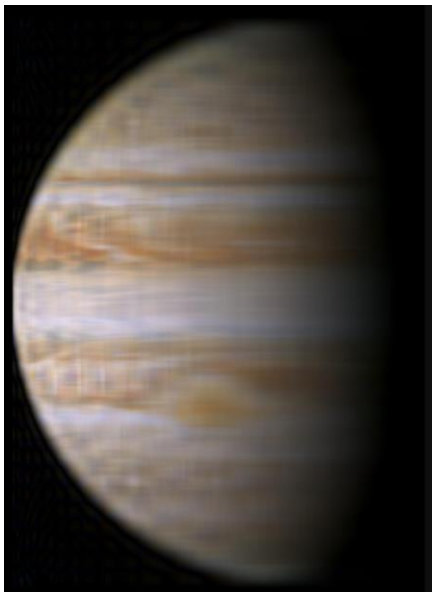
110 start_time = time.time()
111
112 q = 1
113 p = 5
114
115
116
117 Ur, Sr, VTr = randomSVD(r,k,q,p)
118 Ug, Sg, VTg = randomSVD(g,k,q,p)
119 Ub, Sb, VTb = randomSVD(b,k,q,p)
120
121
122 k = round(((100-k)*m*n)/(100*(m+1+n)))
123
124
125 # menggabungkan tiap komponen svd sebanyak k kolom/baris
126 rr = np.dot(Ur[:, :k], np.dot(np.diag(Sr[:k]), VTr[:k, :]))
127 rg = np.dot(Ug[:, :k], np.dot(np.diag(Sg[:k]), VTg[:k, :]))
128 rb = np.dot(Ub[:, :k], np.dot(np.diag(Sb[:k]), VTb[:k, :]))
129
130 print("arranging...")
131
132 # buat array kosong untuk menampung matriks gambar kompresi
133 rimg = np.zeros(img.shape)
134
135 # menggabungkan tiap channel RGB
136 rimg[:, :, 0] = rr
137 rimg[:, :, 1] = rg
138 rimg[:, :, 2] = rb
139
140 # konversi angka matriks
141 for ind1, row in enumerate(rimg):
142     for ind2, col in enumerate(row):
143         for ind3, value in enumerate(col):
144             if value < 0:
145                 rimg[ind1, ind2, ind3] = abs(value)
146             if value > 255:
147                 rimg[ind1, ind2, ind3] = 255
148
149 # konversi gambar & save file
150 compressed_image = rimg.astype(np.uint8)
151 compressed_image = Image.fromarray(compressed_image)
152
153
154 end = time.time()
155
156 # total time taken
157 print(f"Runtime of the program is {end - start_time}")
158
159 return compressed_image
160



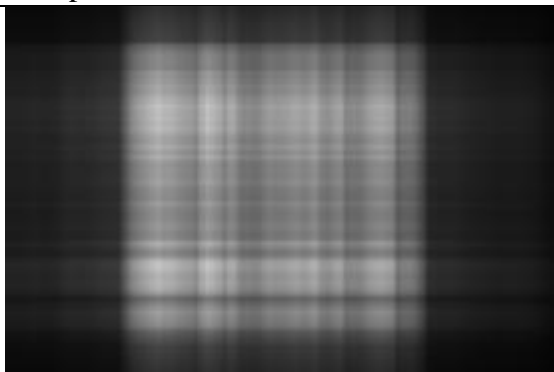
```


BAB 4: EKSPERIMEN

Gambar Asli	Hasil Kompresi
	
	Kompresi 50%
	
	Kompresi 99%

Gambar Asli	Hasil Kompresi
	
	Kompresi 50%
	
	Kompresi 99%

Gambar Asli	Hasil Kompresi
	 <p data-bbox="824 861 1015 892">Kompresi 50%</p>
	 <p data-bbox="824 1522 1015 1554">Kompresi 99%</p>

Gambar Asli	Hasil Kompresi
	
	<p data-bbox="823 596 1015 632">Kompresi 50%</p>  <p data-bbox="823 1005 1015 1041">Kompresi 99%</p>

BAB 5: PENUTUP

5.1 Kesimpulan

Salah satu penerapan Singular Value Decomposition dalam kehidupan nyata adalah kompresi gambar. Singular Value Decomposition mendekomposisi matrix $m \times n$ menjadi 3 matrix yakni, matrix ortogonal $m \times m$, matrix singular $m \times n$, dan matrix transpose orthogonal $n \times n$. Matrix-marix orthognal terdiri atas vektor-verktor eigen matrix yang tersusun dari yang memiliki nilai eigen terbesar hingga terkecil. Untuk mencari vektor eigen dan nilai eigen dapat digunakan algoritma power iteration. Karena power iteration merupakan algoritma yang mencari nilai eigen dan vektor eigen terbesar terlebih dahulu, maka untuk membuat matrix orthogonal yang diperlukan dalam SVD kita dapat menyusun hasil dari Power Iteration.

5.2 Saran

Tim penulis memiliki beberapa saran mengenai pengembangan program ini, antara lain:

1. Optimalisasi algoritma yang digunakan. Algoritma SVD utama yang digunakan dalam program ini, Power Iteration, masih belum secepat algoritma SVD standar bawaan python (Golub-Kahan).
2. Memperbaiki tampilan antarmuka program sehingga lebih mudah digunakan khalayak umum.
3. Program dapat dipublikasikan ke khalayak umum supaya dapat dirasakan kebermanfaatannya dan mendorong terjadinya kolaborasi yang membangun dalam pengembangan program kedepannya.

5.3 Refleksi

Untuk kedepannya masih ada beberapa hal yang dapat kami perbaiki, termasuk kecepatan algoritma, dan juga tampilan website tugas kami. Kemudian kami juga berharap untuk dapat memperbaiki tampilan website sehingga lebih baik dan dapat di-deploy secara online menggunakan tools-tools tertentu.

Kami juga ingin sehingga *image* yang dicompress dapat diambil dari berbagai macam directory tidak hanya terbatas pada directory dimana algoritma backend dan SVD tersebut disimpan.

REFERENSI

Howard Anton, *Elementary Linear Algebra*, 10th edition, John Wiley and Sons, 2010

Hogben, Leslie, ed. *Handbook of linear algebra*. CRC press, 2006.

http://www.math.iit.edu/~fass/477577_Chapter_12.pdf